UNIVERSITATEA „ȘTEFAN CEL MARE” SUCEAVA

FACULTATEA DE INGINERIE ELECTRCĂ ȘI ȘTIINȚA CALCULATOARELOR

PROIECT POO

SERVICE-AUTO MANAGEMENT

COORDONATOR: s.l.dr.ing. Remus Catalin PRODAN

STUDENT: Iroftei Dumitru-Andrei

GRUPA 3123A

CUPRINS

Introducere.....................................................................................................pagina 3

Implementare.................................................................................................pagina 3-5

Explicatie Cod...............................................................................................pagina 5-8

Concluzii.......................................................................................................pagina 8

Bibliografie...................................................................................................pagina 9

INTRODUCERE

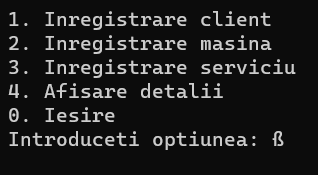
Proiectul reprezinta simularea unei aplicatii de management a unui service auto, utilizand o interfata grafica de tip consola, ce este folosita pentru a usura munca angajatilor in service. Datele introduse vor fi stocate, iar angajatii vor putea identifica cu usurinta clientii.

Limbajul de programare folosit este C++ in mediul de dezvoltare Microsoft Visual-Studio. În realizarea acestui proiect am folosit funcții si biblioteci de baza ale limbajului C++. În realizarea programului am folosit funcții de cautare, adaugare, inregistrare și afișare.

Scopul acestui proiect este ilustrarea avantajului utilizarii paradigmei de programare orientata pe obiecte într-o aplicatie ce necesita gruparea variabilelor, a structurilor de date in anumite categorii.

Proiectul realizat contine 4 clase și anume clasa Client, clasa Masina, clasa Servicii și clasa Principala a Programului AplicatieServiceAuto. În clasa principala a proiectului se aplica procedeul de mostenire al claselor Client, Masina și Servicii, astfel că funcțiile principale pe care se bazeaza aplicatia se afla in aceasta și care face legatura intre clase si fisierele text in care sunt stocate datele.

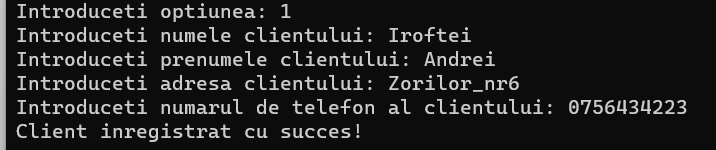
Implementare



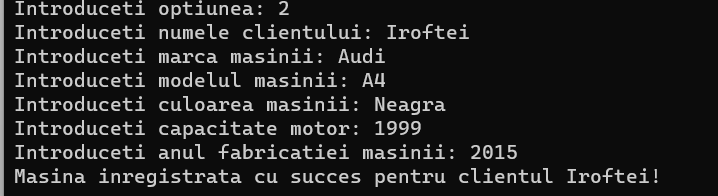
Proiectul prezintă un meniu CMD contextual, în care prin apăsarea unui buton de la

tastatură, pot fi alese diverse opțiuni, precum:

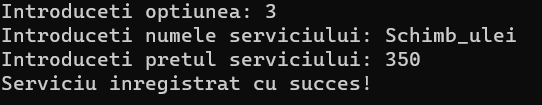
Inregistrare Client:



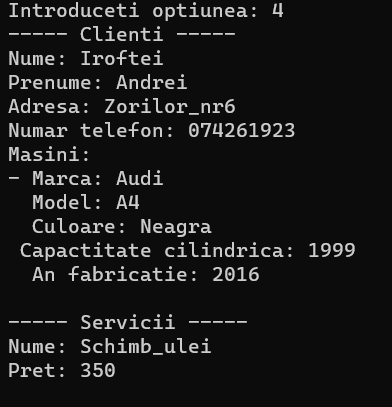
Inregistrare masina:



Inregistrare Serviciu:



Afisare Date Inregistrate:



Dupa folosirea acestor optiuni, clientul este inregistrat si stocat pentru a putea fi contactat dupa terminarea serviciilor dorite.

Explicatii cod

Metodă `inregistrareMasina()` are rolul de a înregistra o mașină pentru un anumit client într-un sistem. Iată ce face:

1. Se solicită introducerea numelui clientului prin intermediul comenzii `cout << "Introduceti numele clientului: ";` și se primește valoarea prin `cin >> nume;`.

2. Se inițializează o variabilă booleană `client\_gasit` cu valoarea `false`. Aceasta va fi utilizată pentru a verifica dacă s-a găsit un client cu numele introdus.

3. Se parcurge lista de clienți (`clienti`) folosind un range-based for loop, unde fiecare element este accesat prin referință cu ajutorul sintaxei `auto& client : clienti`.

4. În interiorul buclei, se verifică dacă numele clientului curent (`client.nume`) coincide cu numele introdus (`nume`).

5. Dacă numele se potrivește, se solicită introducerea informațiilor despre mașină, inclusiv marca, modelul, culoarea, capacitatea cilindrică și anul fabricației. Valorile sunt citite de la tastatură cu ajutorul comenzilor `cin >>`.

6. Se creează un obiect de tip `Masina` (`m`) cu valorile introduse.

7. Se adaugă mașina în lista de mașini a clientului folosind metoda `adaugaMasina()`.

8. Se afișează un mesaj de confirmare a înregistrării mașinii pentru clientul respectiv: `cout << "Masina inregistrata cu succes pentru clientul " << client.nume << "!\n";`.

9. Variabila `client\_gasit` este setată la `true`.

10. Buclei se aplică `break` pentru a ieși din buclă, deoarece nu mai este nevoie de parcurgerea ulterioară a listei de clienți.

În concluzie, această metodă caută un client într-o listă de clienți, pe baza numelui introdus, și dacă găsește un client corespunzător, adaugă o nouă mașină în lista sa de mașini.

Metoda Inregistrare Client:

Această metodă `inregistrareClient()` are rolul de a înregistra un client într-un sistem. Iată ce face:

1. Se declară variabilele de tip `string` `nume`, `prenume`, `adresa` și `numar\_telefon` pentru a stoca informațiile despre client.

2. Se solicită introducerea numelui clientului prin intermediul comenzii `cout << "Introduceti numele clientului: ";` și se primește valoarea prin `cin >> nume;`.

3. Se solicită introducerea prenumelui clientului prin intermediul comenzii `cout << "Introduceti prenumele clientului: ";` și se primește valoarea prin `cin >> prenume;`.

4. Se solicită introducerea adresei clientului prin intermediul comenzii `cout << "Introduceti adresa clientului: ";` și se primește valoarea prin `cin >> adresa;`.

5. Se solicită introducerea numărului de telefon al clientului prin intermediul comenzii `cout << "Introduceti numarul de telefon al clientului: ";` și se primește valoarea prin `cin >> numar\_telefon;`.

6. Se creează un obiect de tip `Client` (`c`) cu valorile introduse pentru nume, prenume, adresa și numărul de telefon.

7. Obiectul `c` este adăugat la sfârșitul listei de clienți (`clienti`) folosind metoda `push\_back()`.

8. Se afișează un mesaj de confirmare a înregistrării clientului: `cout << "Client inregistrat cu succes!\n";`.

Astfel, această metodă solicită informații despre un client și le folosește pentru a crea un obiect `Client`, pe care îl adaugă în lista de clienți.

Inregistrare Serviciu:

Această metodă `inregistrareServiciu()` are rolul de a înregistra un serviciu într-un sistem. Iată ce face:

1. Se declară variabilele de tip `string` `nume` și `float` `pret` pentru a stoca informațiile despre serviciu.

2. Se solicită introducerea numelui serviciului prin intermediul comenzii `cout << "Introduceti numele serviciului: ";` și se primește valoarea prin `cin >> nume;`.

3. Se solicită introducerea prețului serviciului prin intermediul comenzii `cout << "Introduceti pretul serviciului: ";` și se primește valoarea prin `cin >> pret;`.

4. Se creează un obiect de tip `Serviciu` (`s`) cu valorile introduse pentru nume și preț.

5. Obiectul `s` este adăugat la sfârșitul listei de servicii (`servicii`) folosind metoda `push\_back()`.

6. Se afișează un mesaj de confirmare a înregistrării serviciului: `cout << "Serviciu inregistrat cu succes!\n";`.

Astfel, această metodă solicită informații despre un serviciu și le folosește pentru a crea un obiect `Serviciu`, pe care îl adaugă în lista de servicii.

Aplicatia creata utilizeaza aspecte din POO precum mostenirile si polimosrfismul. Aceste aspecte sunt prezentate putin mai jos.

1. Moștenire:

Clasa `Masina`, `Serviciu` și `Client` sunt clasele de bază, iar clasa `AplicatieServiceAuto` este clasa derivată care le moștenește.

Clasa `AplicatieServiceAuto` moștenește membrii și metodele clasei de bază, cum ar fi vectorii `clienti` și `servicii`.

Aceasta permite clasei derivate să acceseze și să utilizeze membrii și metodele clasei de bază.

2. Polimorfism:

Metoda `adaugaMasina()` din clasa `Client` este un exemplu de polimorfism. Aceasta permite adăugarea unui obiect de tip `Masina` la vectorul `masini` al clientului.

Utilizarea polimorfismului permite tratamentul obiectelor derivate (`Masina`) ca și cum ar fi obiecte ale clasei de bază (`Client`) în cadrul metodei `adaugaMasina()`.

Prin urmare, se poate adăuga o mașină la lista de mașini a unui client în cadrul metodei `inregistrareMasina()`.

Astfel, prin moștenire și polimorfism, codul beneficiază de o structură ierarhică și permite utilizarea și manipularea obiectelor într-un mod flexibil și extensibil.

CONCLUZII

Aplicatia ServiceAuto, construită prin utilizarea paradigmei de programare orientate pe

obiecte, folosește mai multe obiecte care comunică prin mesaje specifice pentru a rezolva diverse cerințe puse la dispoziția utilizatorului. Aceste **obiecte** sunt capabile să trimită mesaje (date), să primească mesaje (date) și să proceseze date.

Prezentarea codului și explicația ulterioară arată că acesta utilizează moștenirea și polimorfismul pentru a crea o aplicație simplă de service auto. Moștenirea permite clasa `AplicatieServiceAuto` să beneficieze de membrii și metodele clasei de bază (`Masina`, `Serviciu`, `Client`), facilitând gestionarea datelor legate de clienți, mașini și servicii. Polimorfismul, exemplificat prin metoda `adaugaMasina()` din clasa `Client`, permite tratarea obiectelor derivate (`Masina`) ca obiecte ale clasei de bază (`Client`), oferind astfel o flexibilitate în manipularea și adăugarea mașinilor pentru un client dat.

În concluzie, utilizarea moștenirii și polimorfismului în acest cod permite o structură ierarhică eficientă și extensibilă, precum și o interacțiune flexibilă între obiectele de diferite tipuri. Aceste concepte sunt esențiale în programarea orientată pe obiecte și facilitează dezvoltarea și gestionarea aplicațiilor complexe.

BIBLIOGRAFIE

<http://apollo.eed.usv.ro/~remus/>

<https://www.tutorialspoint.com/cplusplus/cpp_object_oriented.htm>

<https://www.tutorialspoint.com/cplusplus/cpp_inheritance.htm>

<https://www.tutorialspoint.com/cplusplus/cpp_polymorphism.htm>

image11